Computational Models Lecture 8, Spring 2009

- Encoding of TMs
- Universal Turing Machines
- The Halting/Acceptance problem
- The Halting/Acceptance problems are undecidable
- Diagonalization
- Computable functions
- The busy beaver function is not computable (not in book)
- Reductions
- Reducing A to B by Mapping reductions

- Sipser's book, 4.1, 4.2, 5.1

Standard Encoding of Turing Machines (Slightly Modified)

This formulation slightly simplifies last week's convention, without sacrifying generality.

- We assume that our TM, M, has one tape, $\Sigma = \{0, 1\}$ and $\Gamma = \{0, 1, \$, \sqcup\}$. For $L \subseteq \{0, 1\}^*$, this is not a restriction.
- The encoding $\langle M \rangle$ of a TM, M, will use a binary alphabet.
- Blocks of 0's will be used as delimiters.
- A set Q with m states will be indicated by m in unary.
- By conventions states will be 1 through m.
- By convention, q_0 is indicated by state 1, $q_a = q_2$ by 11, and $q_r = q_3$ by 111 (& delimiters!).
- Finally, the transition function δ is encoded as a list of 5-tuples with correct size and no duplications in q, γ entries.
- Important (and Easy): An algorithm (TM) can check that a given string is legal encoding of (any other) TM.

Universal Turing Machine

Universal Turing Machines

We now define the universal Turing machine, U.

On input $\langle M, w \rangle$, where M is a TM and a string w

1. Checks that $\langle M, w \rangle$ is a proper encoding of a TM, followed by a string from Σ^*.
2. Simulates M on input w (some details in next slide)
3. If M on input w enters its accept state, U accept, and if M on input w ever enters its reject state, U reject.

Notice that as a consequence, if M on input w enters an infinite loop, so does U on input $\langle M, w \rangle$.

Univeral Turing Machines: Some Simulation Details

The simulated TM, M, has one tape, has $\Sigma = \{0, 1\}$ and $\Gamma = \{0, 1, \$, \sqcup\}$.

To make life easier, the universal Turing machine, U, that will simulate M, will have several tapes (say five), and a larger work alphabet, Γ'.

On input $\langle M, w \rangle$, where M is a TM and $w \in \{0, 1, \}$ is a string

- Tape 2 of U is the "program tape". The contents of tape 2 will follow the contents of M's single tape, step by step.
- Tape 3 is the "simulation tape". Tape 4 is the "state tape". Tape 5 is the "scratch tape".
- U copies $\langle M \rangle$ to tape 2, and w to tape 3. It places the tape 3 head on the leftmost character of w.

Univeral Turing Machines: Some Simulation Details

- U copies both the the state q (a string of 1s) from tape 4, and the letter it sees on tape 3, $a \in \Gamma$, to tape 5 (with delimiters).
- U compares q, a to the entries in the transition function portion of tape 2.
- Once U finds a match, it updates the state tape (tape 4), writes a letter in the simulation tape (tape 3), and moves the head on tape 3 left/right accordingly.
- If M on input w enters its accept state q_2, U accepts $\langle M, w\rangle$, and if M on input w ever enters its reject state q_3, U rejects $\langle M, w\rangle$.

Notice that as a consequence, if M on input w enters an infinite loop, so does U on input $\langle M, w\rangle$.

Universal Turing Machines (4)

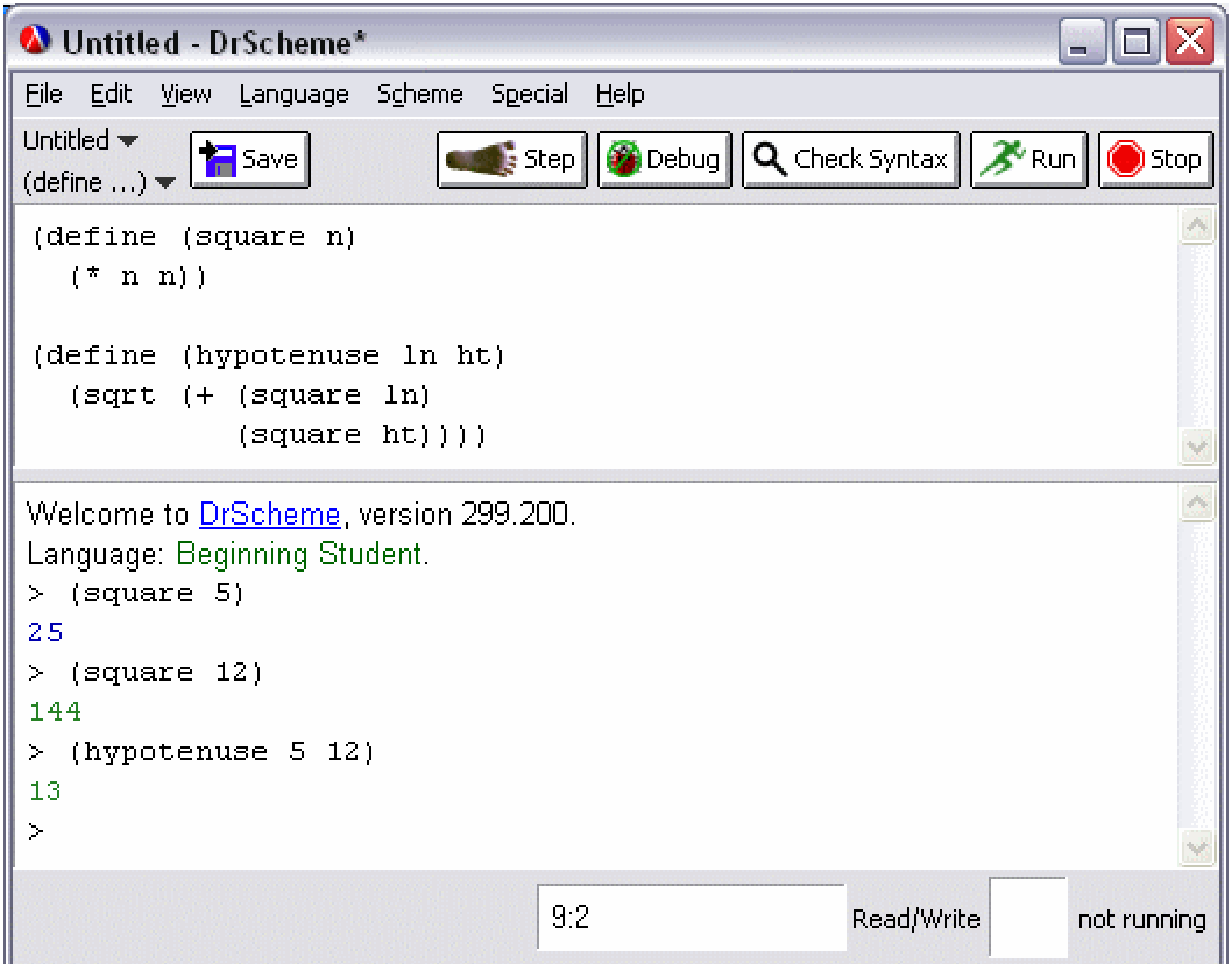
- The universal machine U uses some extra dotting and crossover marks (larger work alphabet) to facilitate comparisons, copying, and erasing.
- The universal machine U obviously has a fixed number of states (100 should do) .
- Despite this, it can simulate machines M with many more states.
- Universal machines inspired the development of stored-program computers in the 40s and 50s.
- Most of you have seen a universal machine, and have even used one!

Universal Turing Machines (5)

- For example, *Dr. Scheme* (interpreter) is a universal Scheme machine.

- It accepts a two part input: “Above the line” – the program (corresponding to $\langle M \rangle$), and “below the line” the input to run it on (corresponding to w).

In case You Forgot (or Repressed)



The Halting/Acceptance Problem

One of the most philosophically important theorems of the theory of computation.

Acceptance Problem: Does a Turing machine accept an input string?

$$A_{\text{TM}} = \{\langle M, w\rangle | M \text{ is a TM that accepts } w\}$$

Halting Problem: Does a Turing machine halt on an input string?

$$H_{\text{TM}} = \{\langle M, w\rangle | M \text{ is a TM and } M \text{ halts on input } w\}$$

Theorem: Both A_{TM} and H_{TM} are undecidable.

The Acceptance Problem

$$A_{\mathsf{TM}} = \{\langle M, w\rangle | M \text{ is a TM that accepts } w\}$$

Before approaching the proof of undecidability, we first notice

Theorem: A_{TM} is recursively enumerable (namely in $\mathcal{RE}$).

Proof: The universal machine accepts A_{TM}. ♣

The Halting Problem

$$H_{\textsf{TM}} = \{\langle M, w\rangle | M \text{ is a TM that halts on the input string } w\}$$

Before approaching the proof of undecidability, we first note

Theorem: $H_{\textsf{TM}}$ is recursively enumerable (namely in $\mathcal{RE}$).

Proof: A slight modification of the universal machine, where the reject state is replaced by the accept state. The modified machine accepts $H_{\textsf{TM}}$. ♣

Acceptance, Again

We are now able to prove the undecidability of

$$A_{\text{TM}} = \{\langle M, w\rangle | M \text{ is a TM that accepts } w\}\,.$$

Proof: By contradiction. Suppose a TM, H, is a decider for A_{TM}.

On input $\langle M, w\rangle$, where M is a TM and w is a string, H halts and accepts if and only if M accepts w. Furthermore, H halts and rejects if M fails to accept w.

Acceptance (2)

On input $\langle M, w \rangle$, where M is a TM and w is a string, H halts and accepts if and only if M accepts w. Furthermore, H halts and rejects if M fails to accept w.

$$H(\langle M, w \rangle) = \begin{cases} accept & \text{if } M \text{ accepts } w \\ reject & \text{if } M \text{ does not accept } w \end{cases}$$

Acceptance (3)

Now we construct a new TM, D, with H as a subroutine.

D does the following

- Calls H to determine what TM, M, does when the input to M is its own description, $\langle M \rangle$.
- When D determines this, it does the opposite.
- So D rejects if M accepts $\langle M \rangle$, and accepts if M does not accept $\langle M \rangle$.

Acceptance (4)

More precisely, D does the following:

- Run H on input $\langle M, \langle M \rangle \rangle$.
- Output the opposite of what H outputs:
  - If H accepts, reject, and
  - If H rejects, accept.

Self Reference (4)

Don't be confused by the notion of running a machine on its own description!

Actually, you should get used to it.

- Notion of self-reference comes up again and again in diverse areas.
- Read "Gödel, Escher, Bach, an Eternal Golden Braid", by Douglas Hofstadter.
- This notion of self-reference is the basic idea behind Gödel's revolutionary result.

Compilers do this all the time . . . .

The Punch Line

So far we have,

$$D(\langle M \rangle) = \begin{cases} \mathit{reject} & \text{if } M \text{ accepts } \langle M \rangle \\ \mathit{accept} & \text{if } M \text{ does not accept } \langle M \rangle \end{cases}$$

What happens if we run D on its own description?

$$D(\langle D \rangle) = \begin{cases} \mathit{reject} & \text{if } D \text{ accepts } \langle D \rangle \\ \mathit{accept} & \text{if } D \text{ does not accept } \langle D \rangle \end{cases}$$

Oh, oh...

Or, more accurately, a contradiction (to what?) ♣

Once Again

- Assume that TM H decides A_{TM}.
- Then use H to build a TM, D, that when given $\langle M\rangle$, accepts exactly when M does not accept.
- Run D on its own description.
- D does:
  - H accepts $\langle M, w\rangle$ when M accepts w.
  - D rejects $\langle M\rangle$ exactly when M accepts $\langle M\rangle$.
  - D rejects $\langle D\rangle$ exactly when D accepts $\langle D\rangle$.
- Last step leads to contradiction.
- Therefore neither TM D nor H can exist.
- So A_{TM} is undecidable!

A Non-enumerable Language

- We already saw a non-decidable language: A_{TM}.
- Can we do better (i.e., worse)?
- Mais, oui!
- We now display a language that isn't even recursively enumerable . . . .

A Non-enumerable Language

Earlier we saw

Theorem: If L and $\overline{L}$ are both enumerable, then L is decidable.

Corollary: If L is not decidable, then either L or $\overline{L}$ is not enumerable.

Definition: A language is co-enumerable if it is the complement of an enumerable language.

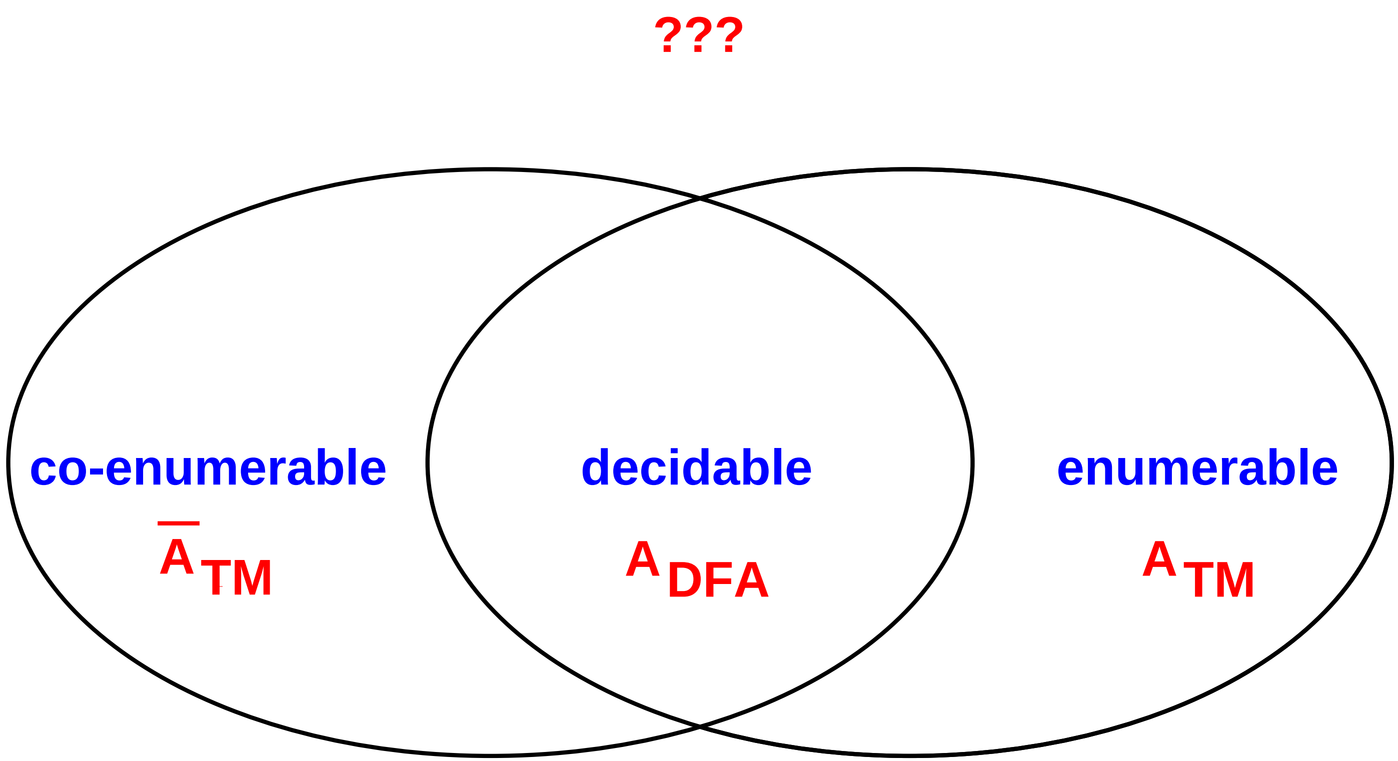
Reformulating theorem Theorem: A language is decidable if and only if it is both enumerable and co-enumerable.

$\overline{A_{\mathsf{TM}}}$ is not Enumerable

Theorem: If L and $\overline{L}$ are both enumerable, then L is decidable.

- We proved that A_{TM} is undecidable.
- On the other hand, we saw that the universal TM, U, accepts A_{TM}.
- Therefore A_{TM} is enumerable.
- If $\overline{A_{\mathsf{TM}}}$ were also enumerable, then by theorem A_{TM} was decidable.
- Therefore $\overline{A_{\mathsf{TM}}}$ is not enumerable. ♣

Languages



Question: Are there any languages in the area marked ???
?

Answer: Yes, heaps (why?)

The Acceptance Problem (again)

We saw

$A_{\text{TM}} = \{\langle M, w\rangle | M \text{ is a TM that accepts } w\}$

Our proof that A_{TM} is undecidable was actually a diagonalization proof.

The Real Numbers

Assume there is a correspondence between $\mathcal{N}$ and $\mathcal{R}$.
Write it down:

n	$f(n)$
1	`3.14159...`
2	`55.55555...`
3	`40.18642...`
4	`15.20601...`

We now show that there is a number x not in this list.

Diagonalization

Pick $0 \leq x \leq 1$, so its significant digits follow decimal point.
Will ensure $x \neq f(n)$ for all n.

n	$f(n)$
1	`3.14159...`
2	`55.55555...`
3	`40.18643...`
4	`15.20607...`

Diagonalization

n	$f(n)$
1	3.14159...
2	55.55555...
3	40.18643...
4	15.20607...

- First fractional digit of $f(1)$ is 1, so pick first fractional digit of x to be something else (say, 2).
- Second fractional digit of $f(2)$ is 5, so pick second fractional digit of x to be something else (say, 6).
- and so on $\ldots$
- $x = 0.2691\ldots$

Diagonalization

A similar proof shows there are languages that are not enumerable.

- the set of Turing machines is countable, but
- the set of languages is uncountable!

- Ergo,
  - there exist languages that are not enumerable (why?)
  - indeed, “most” languages are not enumerable (explain)

$\exists$ Countably Many Turing Machines

Claim: The set of strings, Σ^*, is countable.

Proof: List strings of length 0, then length 1, then 2, and so on. This exhausts all of Σ^*.

The union of countably many finite sets is countable.

$\exists$ Countably Many Turing Machines (2)

Claim: The set of all Turing machines is countable.

Proof: Each TM M has an encoding as a string $\langle M \rangle$. Therefore there is a one-to-one mapping from the set of all TMs into (but not onto) Σ^*.

Since Σ^* is countable, so is the set of all TMs.

The Set of All Languages is Uncountable

Let $\mathcal{B}$ be the set of of infinite binary sequences.

Claim $\mathcal{B}$ is uncountable.

Proof Diagonalization argument, essentially identical to the proof that $\mathcal{R}$ is uncountable.

(additional helpful clue: think of binary sequence as binary expansion!)

The Set of Languages is Uncountable (2)

Let $\mathcal{L}$ be the set of all languages over alphabet Σ.
Recall $\mathcal{B}$ is the set of of infinite binary sequences.
We give a correspondence

$$\chi : \mathcal{L} \to \mathcal{B}$$

called the language's *characteristic sequence*.

- Let $\Sigma^* = \{s_1, s_2, s_3, \ldots\}$ (in lexicographic order).
- Each language $L \in \mathcal{L}$ is associated with a unique sequence $\chi(L) \in \mathcal{B}$:
- the i-th bit of $\chi(L)$ is 1 if and only if $s_i \in L$.

The Set of Languages is Uncountable (3)

Each language $L \in \mathcal{L}$ has a unique sequence $\chi(L) \in \mathcal{B}$: the i-th bit of $\chi(L)$ is 1 if and only if $s_i \in L$.

Example:

$$\begin{array}{llllllllll}
\Sigma^* & \{\varepsilon, & 0, & 1, & 00, & 01, & 10, & 11, & 000 & \ldots\} \\
A & \{ & 0, & & 00, & 01, & & & 000 & \ldots\} \\
\chi(A) & \{0, & 1, & 0 & 1, & 1, & 0, & 0, & 1 & \ldots\}
\end{array}$$

The map $\chi : \mathcal{L} \to \mathcal{B}$

- is one-to-one and onto (why?),
- and is hence a correspondence.
- It follows that $\mathcal{L}$ is uncountable.

TMs vs. Languages

We saw that the set of all Turing machines is countable.

We saw that the set $\mathcal{L}$ of all languages over alphabet Σ is uncountable.

Therefore there are languages that are not accepted by any TM.

This is an existential proof – it does not explicitly show any such language.

Reflections on Diagonalization

This proof that the acceptance problem is undecidable is actually diagonalization in transparent disguise.
To unveil this, let's start by making a table.

	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	$\ldots$
M_1	accept		accept		
M_2	accept	accept	accept	accept	
M_3					
M_4	accept	accept			
$\vdots$					

Entry (i, j) is accept if M_i accepts $\langle M_j \rangle$, and blank if M_i rejects or loops on $\langle M_j \rangle$.

Diagonalization (2)

	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	$\dots$
M_1	accept		accept		
M_2	accept	accept	accept	accept	
M_3					
M_4	accept	accept			
$\vdots$					

Run H on on corresponding inputs. In new table, entry (i, j) states whether H accepts $\langle M_i, \langle M_j \rangle \rangle$.

	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	$\dots$
M_1	accept	reject	accept	reject	
M_2	accept	accept	accept	accept	
M_3	reject	reject	reject	reject	
M_4	accept	accept	reject	reject	
$\vdots$		$\vdots$	$\vdots$	$\vdots$	

Diagonalization (3)

Now we add D to the table.

- By assumption, H is a TM, and therefore so is D.
- D occurs on the list $M_1, M_2, \ldots$ of all TMs.
- D computes the opposite of the diagonal entries.
- At diagonal entry, D computes its own opposite!

	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\ldots$	$\langle D \rangle$
M_1	$\boxed{\text{accept}}$	reject	accept		
M_2	accept	$\boxed{\text{accept}}$	accept		
M_3	reject	reject	$\boxed{\text{reject}}$		
M_4	accept	accept	reject		
$\vdots$		$\vdots$		$\ddots$	
D	reject	reject	accept		$\boxed{???}$
$\vdots$		$\vdots$		$\ldots$	

♣

Halting vs Acceptance Problem

We have already established that A_{TM} is undecidable.

Halting is a closely related problem.

$$H_{\text{TM}} = \{\langle M, w\rangle | M \text{ is a TM and } M \text{ halts on input } w\}$$

Clarification: How does H_{TM} differ from A_{TM}?

Undecidable Problems

$$H_{\text{TM}} = \{\langle M, w\rangle | M \text{ is a TM and } M \text{ halts on input } w\}$$

Theorem: H_{TM} is undecidable.

Proof idea:

- Again, proof by diagonalization.
- Will do this on the blackboard.
- Next week will prove this differently, by a reduction from A_{TM}.

Computable Functions

A TM computes a total function

$$f : \Sigma^* \longrightarrow \Sigma^*$$

if the TM

- when starting with an input w,
- always halts with only $f(w)$ written on tape.

The definition can be extended to functions of more than one variable, where some special separator symbol indicates end of one variable and beginning of next.

Sometimes we have a separate output tape where $f(w)$ is written. This is more convenient, but otherwise makes no difference.

Computable Functions

A TM computes a partial function

$$f : \Sigma^* \longrightarrow (\Sigma^* \cup \perp)$$

if the TM

- when starting with an input w,
- if $f(w)$ is defined, TM halts with only $f(w)$ on tape,
- if $f(w)$ is undefined, TM does not halt.

Computable functions are also called (total or partial) recursive functions.

Computable Functions

Claim: All the "usual" arithmetic functions on integers are computable.

These include addition, subtraction, multiplication, division (quotient and remainder), exponentiation, roots (to a specified precision), modular exponentiation, greatest common divisor.

Even non-arithmetic functions, like logarithms and trigonometric functions, can be computed (to a specified precision), using Taylor expansion or other numeric mathematic techniques.

Exercise: Design a TM that on input $\langle m, n\rangle$, halts with $\langle m + n\rangle$ on tape.

Computable Functions

A useful class of functions modifies TM descriptions.

For example:

On input w:

- if $w = \langle M \rangle$ for some TM,
  - construct $\langle M' \rangle$, where
  - $L(M') = L(M)$, but
  - M' never tries to move off LHS of tape.
- otherwise write ε and halt.

A Non-Computable Function: The Busy Beaver

(taken from
`http://www.saltine.org/joebeaver1.jpg`)

A Non-Computable Function: The Busy Beaver

- We look at all one tape TMs with $\Sigma = \{0, 1\}$ and $\Gamma = \{0, 1, \$, \sqcup\}$.
- Consider the set S_n of all such TMs that have n states and halt on the empty input, ε.
- The set S_n is clearly finite. By definition, if $M \in S_n$ then M on ε runs for finitely many steps.
- Define $BB(n) =$maximum number of steps taken by machines in S_n on the input ε.
- By the discussion above, $BB(n)$ is a well defined, total function.

Theorem: The busy beaver function is not computable.

Proof: On board.

Reducibility

Example:

- Finding your way around a new city
- reduces to ...
- obtaining a city map.

Reducibility, In Our Context

Always involves two problems, A and B.

Desired Property: If A reduces to B, then any solution of B can be used to find a solution of A.

Remark: This property says nothing about solving A by itself or B by itself.

Examples

Reductions:

- Traveling from Boshton to Paris . . .
- reduces to buying plane ticket . . .
- which reduces to earning the money for that ticket . . .
- which reduces to finding a job
  (or getting the $s from mom and dad. . .)

Examples

Reductions:

- Measuring area of rectangle . . .
- reduces to measuring lengths of sides.

Also:

- Solving a system of linear equations . . .
- reduces to inverting a matrix.

Reducibility

If A is reducible to B, then

- A cannot be harder than B
- if B is decidable, so is A.
- if A is undecidable and reducible to B, then B is undecidable.

Additional Undecidable Problems

We have already established that A_{TM} is undecidable.

Here is a related problem – the original halting problem (of Shoshana and Uri :-).

$$H_{\mathsf{TM}} = \{\langle M, w\rangle | M \text{ is a TM and } M \text{ halts on input } w\}$$

Clarification: How does H_{TM} differ from A_{TM}?

Undecidable Problems

$$H_{\text{TM}} = \{\langle M, w\rangle | M \text{ is a TM and } M \text{ halts on input } w\}$$

Theorem: H_{TM} is undecidable.

Proof idea:

- By contradiction.
- Assume H_{TM} is decidable.
- Let R be a TM that decides H_{TM}.
- Use R to construct S, a TM that decides A_{TM}.
- So A_{TM} is reduced to H_{TM}.
- Since A_{TM} is undecidable, so is H_{TM}.

Undecidable Problems

Theorem: H_{TM} is undecidable.

Proof: Assume, by way of contradiction, that TM R decides H_{TM}. Define a new TM, S, as follows:

- On input $\langle M, w \rangle$,
- run R on $\langle M, w \rangle$.
- If R rejects, reject.
- If R accepts (meaning M halts on w), simulate M on w until it halts (namely run U on $\langle M, w \rangle$).
- If M accepted, accept; otherwise reject.

- TM S decides A_{TM}, a contradiction ♣

Undecidable Problems

$$H_{\text{TM}} = \{\langle M, w\rangle | M \text{ is a TM and } M \text{ halts on input } w\}$$

Theorem: H_{TM} is undecidable.

What we actually did was a reduction
from A_{TM} to H_{TM}.

This will be formalized later.

Undecidable Problems (2)

Does a TM accept any string at all?

$$\text{EMPTY}_{\text{TM}} = \{\langle M \rangle | M \text{ is a TM and } L(M) = \emptyset\}$$

Theorem: EMPTY_{TM} is undecidable.

Proof structure:

- By contradiction.
- Assume EMPTY_{TM} is decidable.
- Let R be a TM that decides EMPTY_{TM}.
- Use R to construct S, a TM that decides A_{TM}.

Undecidable Problems (2)

$$\text{EMPTY}_{\text{TM}} = \{\langle M \rangle | M \text{ is a TM and } L(M) = \emptyset\}$$

First attempt: When S receives input $\langle M, w \rangle$, it calls R with input $\langle M \rangle$.

- If R accepts, then reject, because M does not accept any string, let alone w.
- But what if R rejects?

Second attempt: Let's modify M.

Undecidable Problems (2)

$$\text{EMPTY}_{\text{TM}} = \{\langle M \rangle | M \text{ is a TM and } L(M) = \emptyset\}$$

Define M_1: on input x,

1. if $x \neq w$, reject.
2. if $x = w$, run M on w and accept if M does.

M_1 either

- accepts just w, or
- accepts nothing.

Undecidable Problems (2)

Machine M_1: on input x,

1. if $x \neq w$, reject.
2. if $x = w$, run M on w and accept if M does.

Question:
Can a TM construct M_1 from M?

Answer:
Easily, because we need only hardwire w, and add a few extra states to perform the "$x = w$?" test.

Undecidable Problems (2)

$$\text{EMPTY}_{\text{TM}} = \{\langle M\rangle | M \text{ is a TM and } L(M) = \emptyset\} \ .$$

Theorem: EMPTY_{TM} is undecidable.

Define S as follows:

On input $\langle M, w\rangle$, where M is a TM and w a string,

- Construct M_1 from M and w.
- Run R on input $\langle M_1\rangle$,
- if R accepts, reject; if R rejects, accept.

- TM S decides A_{TM}, a contradiction ♣

Undecidable Problems (3)

Does a TM accept a regular language?

$$\text{REG}_{\text{TM}} = \{\langle M \rangle | M \text{ is a TM and } L(M) \text{ is regular}\}$$

Theorem:
REG_{TM} is undecidable.

Skeleton of Proof:

- By contradiction.
- Assume REG_{TM} is decidable.
- Let R be a TM that decides REG_{TM}.
- Use R to construct S, a TM that decides A_{TM}.

But how?

Undecidable Problems (3)

$$\text{REG}_{\text{TM}} = \{\langle M \rangle | M \text{ is a TM and } L(M) \text{ is regular}\}$$

Intuition: Modify M so that the resulting TM accepts a regular language if and only if M accepts w.

Design M_2 so that

- if M does not accept w, then M_2 accepts $\{0^n 1^n | n \geq 0\}$ (non-regular)
- if M accepts w, then M_2 accepts Σ^* (regular).

Undecidable Problems (3)

Given M and w, construct M_2:

On input x,

1. If x has the form $0^n 1^n$, accept it.
2. Otherwise, run M on input w and accept x if M accepts w.

Claim:

- If M does not accept w, then M_2 accepts $\{0^n 1^n | n \geq 0\}$.
- If M accepts w, then M_2 accepts Σ^*.
- The function: On input $\langle M, w \rangle$, output $\langle M_2 \rangle$, is computable.

Undecidable Problems (3)

$$\text{REG}_{\text{TM}} = \{\langle M \rangle | M \text{ is a TM and } L(M) \text{ is regular}\}$$

Theorem: REG_{TM} is undecidable.

Define S:

On input $\langle M, w \rangle$,

1. Construct M_2 from M and w.
2. Run R on input $\langle M_2 \rangle$.
3. If R accepts, accept; if R rejects, reject.

- TM S decides A_{TM}, a contradiction ♣

Undecidable Problems (4)

Are two TMs equivalent?

$$\text{EQ}_{\text{TM}} = \{\langle M_1, M_2\rangle \quad | \quad M_1, M_2 \text{ are TMs and } L(M_1) = L(M_2)\}$$

Theorem: EQ_{TM} is undecidable.

We are getting tired of reducing A_{TM} to everything.

Let's try instead a reduction from EMPTY_{TM} to EQ_{TM}.

Undecidable Problems (4)

$$\mathrm{EQ}_{\mathrm{TM}} = \{\langle M_1, M_2 \rangle \quad | \quad M_1, M_2 \text{ are TMs and } L(M_1) = L(M_2)\}$$

Theorem: $\mathrm{EQ}_{\mathrm{TM}}$ is undecidable.
Idea:

- $\mathrm{EMPTY}_{\mathrm{TM}}$ is the problem of testing whether a TM language is empty.
- $\mathrm{EQ}_{\mathrm{TM}}$ is the problem of testing whether two TM languages are the same.
- If one of these two TM languages happens to be empty, then we are back to $\mathrm{EMPTY}_{\mathrm{TM}}$.
- So $\mathrm{EMPTY}_{\mathrm{TM}}$ is a special case of $\mathrm{EQ}_{\mathrm{TM}}$.

The rest is easy.

Undecidable Problems (4)

$$\text{EQ}_{\text{TM}} = \{\langle M_1, M_2\rangle \quad | \quad M_1, M_2 \text{ are TMs and } L(M_1) = L(M_2)\}$$

Theorem: EQ_{TM} is undecidable.

Let M_{NO} be this TM: On input x, reject.

Let R decide EQ_{TM}.

Let S be: On input $\langle M\rangle$:

1. Run R on input $\langle M, M_{\text{NO}}\rangle$.
2. If R accepts, accept; if R rejects, reject.

If R decides EQ_{TM}, then S decides EMPTY_{TM}. ♣

Bucket of Undecidable Problems

Same techniques prove undecidability of

- Does a TM accept a decidable language?
- Does a TM accept a context-free language?
- Does a TM accept a finite language?
- Does a TM halt on all inputs?
- Is there an input string that causes a TM to traverse all its states?

Reducibility

So far, we have seen many examples of reductions from one language to another, but the notion was neither defined nor treated formally.

Reductions play an important role in

- decidability theory (here and now)
- complexity theory (to come)

Time to get formal.

Mapping Reductions

Definition: Let A and B be two languages. We say that there is a mapping reduction from A to B, and denote

$$A \ \leq_m B$$

if there is a computable function

$$f : \Sigma^* \longrightarrow \Sigma^*$$

such that, for every w,

$$w \in A \Longleftrightarrow f(w) \in B.$$

The function f is called the reduction from A to B.

Missing Figure Here

A mapping reduction converts questions about membership in A to membership in B

Mapping Reductions

Theorem:

If $A \leq_m B$ and B is decidable, then A is decidable.

Proof: Let

- M be the decider for B, and
- f the reduction from A to B.

Define N: On input w

1. compute $f(w)$
2. run M on input $f(w)$ and output whatever M outputs.

Mapping Reductions

Corollary: If $A \leq_m B$ and A is undecidable, then B is undecidable.

In fact, this has been our principal tool for proving undecidability of languages other than A_{TM}.